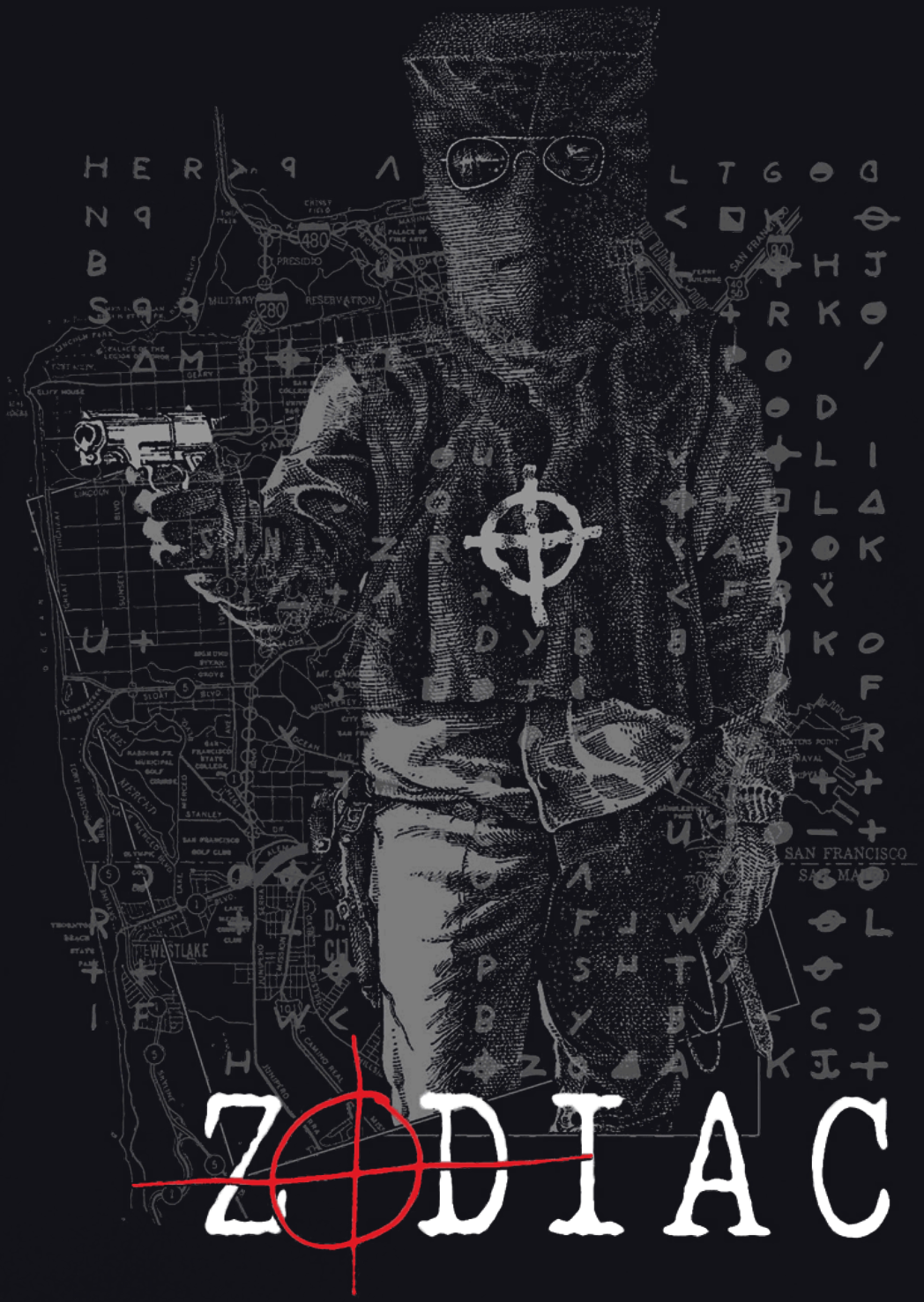
دفترچه راهنما پیکار Zodiac



مقدمه

این دفترچه برای پیکار استعدادیابی Zodiac طراحی شده است. پیکار ۱۰ مرداد ساعت ۲۱:۰۰ شروع میشه و تا ۱۳ مرداد ساعت ۲۱:۰۰ شب ادامه داره (پیکار برای یه هفته بالا می‌مونه که شما بتونین تمرین کنین). جواب پیکار تا ۱۹ مرداد اعلام می‌شه (نفراتی که برنده بشن باشون تماس گرفته میشه و اگه توی کلاس اوسپ زیرو ثبت‌نام کرده باشن، وجهشون عودت داده میشه.). این پیکار برای افرادی که **«توی امنیت صفرن»** کاملا مناسبه، البته که شما نیاز دارین ۳ روز خوب وقت بذارین و کلی سرچ کنین، کلی گیر کنین و سر و کله بزنین ولی مطمئنیم که می‌تونین حل کنین. یکی از اهداف مهم این پیکار **«گسترش علم»** هست. بدیهیه که توی ۳ روز نمیشه به یه هکر تبدیل شد ولی هکرهای خیلی خفن هم بعضا از الگوهای ساده‌ای استفاده می‌کنن. مهم تفکر یا ماییندست یه هکره که سعی کردیم روحش توی این پیکار دمیده بشه. ما (تیم هانت Voorivex) امسال با خیلی از همین این تکنیک‌های ساده کلی آسیب‌پذیری کشف کردیم و مبلغ قابل توجهی بانتی گرفتیم. امیدوارم که تجربه خوبی از شرکت در این پیکار بدست بیارین.

مرحله ۰

بچه‌ها برای حل سوالات شما باید یه دانش پایه از وب داشته باشین، البته که یاد گرفتن همین مبانی ساده ممکنه زمان‌بر باشه و طول بکشه. پس نباید فعلا خیلی توش عمیق بشیم (در زمان مناسب که حتما) و نباید هم خیلی سطحی نگاه کنیم، یه سری قسمت‌های کاربردیشو دیگه باید بلد باشیم. برای اینکه به این حالت برسین دوره امنیت اپلیکیشن رو پیشنهاد می‌کنم (یادتون باشه توی این دفترچه یه سری اصطلاحات بکار رفته که برای درک بهترش نیاز دارین ویدیوها رو ببینین). می‌تونین روی 2x ببینین که تقریبا ۶ ساعت وقتتون رو می‌گیره و تقریبا **علم پایه‌ای** که باید برای حل پیکارها بدونین رو بدست میارین. البته که برای حل یک سری از پیکارهای استعدادیابی ممکنه نیاز به مطالعه بیشتری پیدا کنین، ولی خب علم پایه توی این سری تا حد قابل قبولی کاور شده.

مرحله ۱

برای حل این مرحله شما باید بدونین URL چیه؟ دقیقا مثل آدرس توی دنیای واقعی عمل می‌کنه. شما برای اینکه به مقصد مشخصی برین نیاز دارین آدرس دقیق رو بدونین، آدرس شامل شهر، منطقه، خیابون، کوچه، پلاک و واحد هست. یعنی دقیقا باید مقصد مشخص باشه. توی وب URL دقیقا میشه همون آدرس. وقتی ما روی یه لینک مثلا لینک عکس کلیک می‌کنیم، مرورگر بصورت خودکار URL رو توی Address Bar عوض می‌کنه و ما به محتوای اون عکس دسترسی پیدا می‌کنیم (که اینجا مرورگر میاد اطلاعات عکس رو ترجمه می‌کنه و ما بصورت بصری اونو می‌بینیم، در اصل یه سری داده یا اطلاعات برای مرورگر فرستاده میشه). آیا بدون اینکه روی لینکی کلیک کنیم، با تغییر URL می‌تونیم محتوای صفحه دیگه رو ببینیم؟ جواب بله است بشرطی که برای اون URL یک محتوا در وب‌سرور یا بصورت کلی در سرور مقصد وجود داشته باشد. مطالعه بیشتر؟ اینجا.

مرحله ۲

وقتی ما یه وب‌سایت رو باز می‌کنیم، در واقع به یک URL درخواست HTTP ارسال می‌کنیم و در جواب کلی کد HTML و JavaScript و CSS دریافت می‌کنیم. مرورگر بعد از دریافت این کدها شروع می‌کنه به تفسیر اونها (لغات تخصصی اینجا parse یا interpret یا render هستن) و در نهایت به ما یه سری صفحات که بش می‌گیم وبسایت رو نشون میده. به علاوه از این کار، ما میتونیم از مرورگر کدهای پشت این وب‌سایت هم بگیریم و ببینیم (که البته اینجا یه تقسیم‌بندی وجود داره، کدهای Client و کدهای Server که الان مقصود ما کدهای Client هست)، یعنی دقیقا اون چیزی که مرورگر اونو تفسیر میکنه. برای دیدن سورس مرورگر کافیه وب‌سایت رو باز کنین، بعد کلیک راست کنین و قسمت View Page Source رو انتخاب کنین.

مرحله ۳

وقتی شما سورس HTML صفحه رو میبینین، در واقع نه همه کدهای Client بلکه بخشی از کدهای Client اون وب‌سایت رو دارین مشاهده می‌کنین. خب شاید بپرسید چرا؟ مگه نه که کل کدهای Client روی مرورگر میاد و مرورگر اونا رو تفسیر میکنه و نشون میده؟ جواب اینه که بله، ولی یه سری از کدها خودشون باعث لود شدن یه سری کد میگه میشن. یعنی یک وب‌سایت میتونه لینک‌های مختلفی از CSS و یا JS داشته باشه که Parse شدنشون باعث لود شدن ULRها یا کدهای دیگه بشه. برای اینکه کلیه کدهای لود شده رو ببینین، کافیه روی صفحه وب‌سایت کلیک راست کنین و Inspect رو انتخاب کنین، بعدش روی تب Debugger می‌تونین کلیه کدهای Client رو بصورت درختی ببینین و روی هر کدوم کلیک کنین داده‌های اون قسمت رو براتون نشون میده.

مرحله ۴

امیدوارم فیلم‌های وب‌سرور رو دیده باشین چون یکی از مهمترین قسمت‌هاست. وب‌سرورها در حالات مختلفی استفاده یا Setup میشن. یکی از سنتی‌ترین راه‌های استفاده از اونا پیاده‌سازی Document Root هست. وب‌سایت‌های قدیمی‌تر معمولا همه از این معماری استفاده می‌کنند. یعنی یه سری مسیر و فایل توی وب‌سرور وجود داره و کار وب‌سرور اینه اینا رو نگاشت کنه روی URL وبسایت. مثلا فرض کنیم Document Root تو سرور آدرس /var/www/html رو داشته باشه (قابل تنظیم هست). فرض کنین توی اون مسیر یه پوشه با نام mamad وجود داره که توش فایل فایل abbas.txt هست. کار وب‌سرور اینه که بیاد این پوشه و فایل رو نگاشت کنه روی آدرس وب‌سایت، فرض کنیم آدرس yasho.com هست، پس ما با این URL می‌تونیم به محتوای فایل دسترسی پیدا کنیم:

https://yasho.com/mamad/ababs.txt

امیدوارم متوجه mapping شده باشین. حالا سوال پیش میاد اگه ما URL رو دستکاری کنیم و بخوایم محتوای پوشه mamad رو ببینیم، وب‌سرور این اجازه رو به ما میده؟ جواب سوال اینه: ممکنه بشه ممکنه

نه و این همش بستگی به تنظیمات وب‌سرور داره. راستی یادمون باشه قرار نیست همه URLهای توی وب‌سایت وجود داشته باشن، ممکنه بعضیا از روی سرور پاک شده باشن و مدیر سایت یادش رفته باشه کدهای HTML رو درست کنه. مطالعه بیشتر؟ اینجا

مرحله ۵

یکی از مهم‌ترین بخش‌های کشف آسیب‌پذیری، تشخیص الگوی مشابه یا Pattern Recognition هست. بذارین مثال واقعی بزنم، توی یه سازمان نام کاربری یک کاربر در سامانه ایمیل m.ahamadi عه، بنظر شما الگویی توی این نام کاربری وجود داره؟ اگه کمی دقت کنیم m ممکنه حرف اول اسم کاربر باشه پس میشه گفت یک الگویی وجود داره. یا فرض کنین توی یک وب‌سایت ما یه URL داریم که آخرش users.htm وجود داره، الگویی توش میبینین؟ مثلا admin.htm یا user.htm چی؟ یا مثال دیگه، فرض کنین آدرس یه وب‌سایت اینه: panel-prod.site.com بنظرتون اینجا چی؟ بنظرتون کلمه prod اشاره به چی میکنه؟ production یا محصول نهایی؟ پس شاید وب‌سایت تستی روی panel-dev.site.com باشه که اینجا dev مخفف development هست.

مرحله ۶

بعضی وقتا کشف الگوها ممکنه خیلی ساده باشه (مرحله قبلی)، ولی بعضی وقتا باید Out of Box فکر کرد، یعنی الگو وجود داره ولی نه اونطوری که مغز ما تصور می‌کنه و باید کمی خلاقیت بخرج بدیم. مثلا عدد بزرگ 23581321345589144 رو در نظر بگیرین، بنظرتون الگویی داره یا ارقام تصادفیه؟ بش فکر کنین و اگه متوجه نشدین عدد رو توی گوگل سرچ کنین.

مرحله ۷

فرم‌های HTML یکی از رابط‌های گرافیکی برای انتقال داده بین کاربر و سرور هستن. این فرم‌ها اینطوری کار میکنن که کاربر اطلاعات خودش رو وارد می‌کنه و معمولا یه دکمه Submit دارن که باعث میشه اطلاعات به سمت سرور ارسال بشن. روش انتقال یا نوع Encoding توی این فرم‌ها بصورت پیشفرض URL Encoded هست، اگه نمیدونین این روش انتقال چیه باید کمی کد بزنین و تمرین کنین (سادس، توی سری «امنیت اپلیکیشن» توی یوتوب توضیح دادم کامل). این فرم‌ها معمولا یه سری Input دارن که کاربر اطلاعات رو توشون وارد می‌کنه، ولی خود این Inputها انواع مختلفی دارن. مثلا دیدین وقتی میخواین پسورد رو توی یه سایت وارد کنین براتون ستاره‌ای نشون میده؟ بخاطر اینه که Input از نوع Password تعریف شده. ما دو نوع Input داریم که تو هک برامون کمی جذاب‌ترن، یکیش نوع hidden و اون یکی disabled هست (مطالعه بیشتر). نوع اول برای ما قابل دیدن نیست (ولی توی سورس‌های HTML هست) و نوع دوم برامون قابل دیدنه اما سمت سرور ارسال نمیشه. اگه ارسال نمیشه چرا هست؟ یا میشه کاری کرد که ارسال بشه؟

یادمون باشه وقتی روی صفحه مرورگر کلیک راست می‌کنیم و Inspect می‌زنیم، هم می‌تونیم کدهای HTML رو ببینیم و هم می‌تونیم اونا رو ادیت یا ویرایش کنیم.

مرحله ۸

جاوااسکریپت یکی از بخش‌های جدا نشدنی از وب‌سایت‌های امروزیه، و چه بخوایم چه نخوایم توی وب هکینگ و هانت هم خیلی برامون کاربرد داره (بیشتر از خیلی). اینجا با یه قابلیت ابتدایی جاوااسکریپت آشنا میشیم: Event Handler. یعنی چی؟ یعنی یک قطعه کد جاوااسکریپ وقتی اجرا میشه که اتفاق خاصی بیفته. مثلا یه ایونت هندلر داریم بنام onmouseover، وقتی که کاربر اشاره‌گر موسش رو روی المان HTML مربوطه ببره که این این ایونت رو داشته باشه، کد جاوااسکریپت داخلش اجرا میشه. یه مثال با کد میزنم:

```
<img src="axe-mamad.jpg" onerror=alert(":(")>
```

این کد رو با نام دلخواه توی یه فایل HTML ذخیره کنین (مثلا test.html) و روش کلیک کنین تا باز شه. بمحض باز شدن یه HTML Element از نوع IMG ساخته میشه که میخواد axe-mamad.jpg رو لود کنه (یه درخواست HTTP زده میشه) ولی چون چنین عکسی وجود نداره، کد داخل ایونت onerror توسط مرورگر اجرا میشه، که این کد روی صفحه براتون یه پیغام می‌نویسه. خب حالا که با این مفهوم آشنا شدین، یک ایونت هندلر رو بتون معرفی می‌کنم: onsubmit. وقتی که یه فرم میخواد Submit بشه این ایونت هندلر فراخوانی میشه. راستی ما دو راه برای اتصال یه ایونت هندلر داریم، راه اول بصورت مستقیم که تو کد بالا دیدین، راه دوم بصورت عمومی از طریق Event Listener توسط کدهای جاوااسکریپت. یعنی با جاوااسکریپت یک Element رو انتخاب می‌کنیم (یا کل رو) و ایونت هندلر رو بش متصل می‌کنیم. خب توضیحات زیاد شد، بعنوان نکته آخر بگم که از طریق Debugger توی مرورگر می‌تونین کدهای JS یک سایت رو هم ویرایش کنین.

مرحله ۹

توی مرحله ۷ یاد گرفتیم که با فرم‌های HTML میشه درخواست HTTP فرستاد. ولی آیا تنها راه ارسال درخواست HTML عه؟ خیر. ارسال و دریافت درخواست‌های HTTP توسط JS از قسمت‌های جدانشدنی وب امروزه. راه‌های مختلفی برای ارسال درخواست هست، یکیش مثلا کتابخونه XmlHttpRequest، و یا مثلا Fetch که خیلی پرطرفداره. توی مرحله قبل یاد گرفتیم توی Debugger مرورگر می‌تونیم فایل‌های JS رو ببینیم و همزمان می‌تونیم کدها رو ویرایش هم بکنیم.

مرحله ۱۰

معمولا وب‌سایت‌ها نیاز دارن یه سری داده رو سمت کاربر ذخیره کنن. خب اولین سوالی که ممکنه پیش بیاد اینه که چرا؟ چون پروتکل HTTP یه پروتکل Stateless عه. دلیل دیگش بحث Performance ای هست، خب سرور هر چی بیشتر داده‌ها و پردازشش رو سمت کاربر انجام بده کمتر دردش می‌گیره. یکی از

جاهایی که محل ذخیره‌سازی اطلاعات هست Cookie عه. دو راه ذخیره‌سازی یا ست کردن کوکی وجود داره، با استفاده از JS و یا هدر Set-Cookie. توی روش دوم مرورگر توی جوابی که از سرور برمی‌گرده دنبال اطلاعات کوکی می‌گرده و اگه چیزی پیدا کنه، تو قسمت Storage ذخیرش می‌کنه.

مرحله ۱۱ و ۱۲

هکرا یا هانترا ابزارهای مختلف و کارهای مختلفی انجام میدن، ولی میشه گفت همشون ترافیک HTTP رو شنود می‌کنن. Capture کردن یا Intercept کردن ترافیک امکان شناسایی و تحلیل وب‌اپلیکیشن رو به ما میده. با نرم‌افزارهای زیادی میشه این کارو کرد، من Burp Suite رو پیشنهاد می‌کنم (نسخه Community).

مرحله ۱۳

از بین هدرهای HTTP، بعضی هدرها از اهمیت بیشتری برخوردارن و کلی آسیب‌پذیری بواسطه اونا کشف شده. از هدرهای مهم میشه به Referer و User Agent اشاره کرد. سایت‌ها معمولا مقادیر این هدرها رو برای آمار توی دیتابیس ذخیره می‌کنن و SQLi می‌خورن. البته همیشه اینطوری نیست و خب خیلیا بصورت امن این ذخیره‌سازی رو انجام میدن. بعضی از سایت‌ها برای چک‌های امنیتی از این هدرها کمک می‌گیرن (مثلا هدر Referer برای جلوگیری از CSRF) و بعضی از سایت‌ها هم با توجه به مقادیر این هدرها، محتوای مختلفی رو به کاربر نشون میدن (این تجربه رو توی باگ‌بانتی خیلی داشتم). یادمون باشه این هدرا تحت کنترل کامل کاربرند.

مرحله ۱۴

راجع به این مرحله توضیحی نمی‌دم که همیشه یادتون بمونه: برای هک باید چشای تیزبینی داشته باشین و به جزئیات دقت کنین.

مرحله ۱۵

یکی از متودهای ساده و همیشه کاربری در وب هکینگ، Verb Tampering هست. ساده بخاطر اینکه ساخت پکت HTTP رو خیلی از ابزارها برامون انجام میدن و نیاز نیست دستی پکت رو درست کنیم، کاربردی بخاطر اینکه این روش هنوز در سال ۲۰۲۴ باعث دور زدن و یا کشف آسیب‌پذیری‌های مختلف میشه. یادمون باشه که ما توی پکت‌های HTTP انواع مختلفی از Encoding Type ها داریم که هر کدوم ممکنه رفتار مختلفی از خودشون نشون بدن.

مرحله ۱۶ و ۱۷

یکی از قدیمی‌ترین یا شاید اولین روش‌های کنترل دسترسی کاربران، احراز هویت از طریق Basic Authentication عه. این روش احراز هویت توسط Web Server صورت می‌گیره، بطوری که وب‌سرور هدر WWW-Authenticate رو برای کاربر می‌فرسته، مرورگر این هِدِر رو بصورت یه باکس بصری برای کاربر نشون میده، باکسی که از کاربر نام کاربری و رمز عبور می‌گیره، کاربر اطلاعات رو وارد می‌کنه و مرورگر در قالب هِدِر Authorization اونا رو برای وب‌سرور می‌فرسته، اینجا وب‌سرور اطلاعات رو صحت‌سنجی میکنه و در صورت درست بودن اطلاعات، دسترسی رو به کاربر میده. راه‌های مختلفی برای حمله به این نوع احراز هویت وجود داره، مثلا خیلی مواقع از نام کاربری و پسورد ساده استفاده شده، یا ممکنه در پیکربندی ضعف امنیتی وجود داشته باشه.

مرحله ۱۸ و ۱۹

مهم‌ترین مفهوم امنیتی مرورگرها SOP عه. میشه گفت پاشنه آشیل امنیت تمام وب‌سایت‌ها به این پالیسی بستگی داره. اگه SOP به درستی توسط مرورگر پیاده‌سازی نشده باشه، امکان خوندن تمامی GMailهای شما در لحظه وجود داره، بدون اینکه GMail بتونه جلوشو بگیره. ولی خیلی وقت‌ها سایت‌ها نیاز دارن منابع خود رو سمت کاربر با همدیگه به اشتراک بذارن و پالیسی SOP از این امر جلوگیری میکنه. چندین راه برای این امر وجود داره، یکیش PostMessage، یکیش JSONP و یکیش هم CORS که این روزا خیلی محبوبه. CORS در واقع میاد قوانین SOP رو دستکاری می‌کنه، البته که غیرفعال کردن کامل SOP منجر به آسیب‌پذیری‌های خطرناکی میشه، پس باید امنیت در حین پیاده‌سازی CORS جدی گرفته بشه. و سوالی که پیش میاد اینه: آیا همیشه CORS رو امن پیاده‌سازی می‌کنن؟

مرحله ۲۰

یادمون باشه همیشه همه راه‌های اخذ داده توسط یه وب‌سایت رو نمی‌تونیم ببینیم، بعضی وقتا ممکنه این ورودی‌ها کاملا و یا تا حدی مخفی باشن.

مرحله ۲۱

چگونگی ذخیره‌سازی اطلاعات کاربران در دیتابیس یکی از قسمت‌های مهم امنیت وب هست. مخصوصا بخش پسوردهای کاربران. کلیه الزامات امنیتی، ذخیره‌سازی پسوردها بصورت Clear Text یا متن خوانا رو منع می‌کنه. دلیل واضحی داره، اگه روزی دیتابیس لو رفت (با یک آسیب‌پذیری و یا نفوذ کامل)، پسوردها قابل دیدن نباشه. حتی می‌تونیم یه مرحله جلوتر بریم، چرا باید مدیر دیتابیس بتونه پسوردهای کاربران رو ببینه؟ ممکنه ممد پسوردش رو جای دیگه هم استفاده کرده باشه. خلاصه که نحوه نگهداری پسورد همیشه جز الزامات امنیتی از جمله OWASP بوده. بهترین راه نگه‌داری پسوردهای کاربران بصورت Hash عه، اونم

نوع قوی‌اش (نه مثلا md5). البته یه مشکلی این وسط وجود داره: سایت‌هایی که کلیه هش‌ها رو محاسبه می‌کنن. یعنی اگه روزی دیتابیس لو بره، رمزهای عبور ساده و یا حتی نسبتا سخت هم ممکنه توسط این سایت‌ها قابل کشف باشن.

مرحله ۲۲

برای جلوگیری از این اتفاق، نه تنها Hash بلکه وجود Salt نیز الزامی هست. Salt یه مقدار تصادفیه که به اول یا آخر هر پسورد اضافه میشه و باعث میشه دیگه یه هش رو نشه به سادگی با دو تا سرچ تو اینترنت پیدا کرد. برای مثال هش e10adc3949ba59abbe56e057f20f883e رو در نظر بگیرین، توی گوگل سرچش کنین و میبینین به سرعت می‌تونین به مقدار اصلیش برسین. اگه این هش پسورد یه کاربر باشه، پسوردش افشا میشه. حالا فکر کنین که وب‌سایت برای این کاربر Salt با مقدار fj30dka رو در نظر می‌گیره. یعنی هش ذخیره شده برای کاربر به b79f1d2c29ce5e9d78c5ea60d7225096 تغییر می‌کنه، حالا این هش رو سرچ کنین، چیزی بدست نمیاد. البته که مقدار هش باید توی دیتابیس ذخیره بشه (وگرنه خود سایت راهی برای تشخیص اصالت پسورد نداره) و در صورت لیک شدن دیتابیس سایت، هکر مجبوره بشینه و آفلاین کرک رو شخصا خودش انجام بده (بازم امن نیست ولی نسبت به هش خالی سخت‌تره). یکی از پسوردلیست‌هایی که هکرا خیلی ازش خوششون میاد پسورد لیست Rockyou هست، اگه نیاز به پسورد لیست پیدا کردین از این لیست کمک بگیرین.

مرحله ۲۳

بعد از هک شدن دیتابیس تپسی یادمه توی شبکه‌های اجتماعی یک سری اشخاص می‌گفتن: «چرا دیتابیس رمزنگاری نشده؟ چرا پسوردها رمز نشده؟» متاسفانه این گزاره درست نیست. درسته که رمزگذاری کردن پسوردها بهتر از هیچیه، ولی قطعا نسبت به هش از امنیت پایین‌تری برخورداره. چرا؟ چون حتما کلید رمزنگاری باید توی خود سایت وجود داشته باشه، یا توی دیتابیس، یا توی سورس‌کد، یا توی Environment و یا جاهای دیگه، پس اگه نفوذی صورت بگیره با احتمال بالا قابل کشف هست و اگه کلید رمزنگاری لو بره، ۱۰۰٪ پسوردها افشا میشه. ولی توی Hashing، هیچ وقت نمیشه با اطمینان گفت ۱۰۰٪ هش‌ها قابل کرک هستن، حتی برای مواقعی که الگوریتم ناامن MD5 اونم بدون Salt استفاده شده باشه. خب حالا همیشه هم نمیشه کلید رمزنگاری رو پیدا کرد (فکر کنین یه SQLi ابتدایی که فقط باش میشه دیتابیس رو دامپ کرد و کلید توی سورس کدهای سایت هست)، اینجور مواقع هکرها به دنبال Brute Force کردن کلید رمزنگاری میرن و سوالی که پیش میاد اینه: آیا وب‌سایت از کلید رمزنگاری امنی استفاده کرده یا کلید قابل کشف در لیست‌های توی اینترنت هست؟

مرحله ۲۴

اغلب خطاهایی که وب اپلیکیشن‌ها نشان می‌دهند حاوی اطلاعات حساسی است. این اطلاعات بعضی وقتا منجر به کشف آسیب‌پذیری می‌شود، بعضی وقت‌ها هم خود اطلاعات انقدر حساسند که افشای اطلاعات محسوب می‌شود (مثل افشای API Key سمت سرور در یک خطا).

مرحله ۲۵

قدمت این آسیب‌پذیری خیلی زیاده، نسبت به قدیم کمتر شده ولی هنوز توی سازمان‌های بزرگ هم پیدا میشه. ابزارهای نسبتا خوبی هم برای تشخیص و اکسپلویت این آسیب‌پذیری وجود داره. من خودم شخصا برای تشخیص SQLi بصورت دستی عمل می‌کنم، ولی برای اکسپلویت از ابزار استفاده می‌کنم. البته که همیشه ابزار نمی‌تونه کامل اکسپلویت کنه و خیلی وقتا مجبور میشم ترافیک رو بازبینی کنم و تغییراتی بدم تا درست کار کنه.

مرحله ۲۶ تا ۳۰

با حل مرحله ۲۵ و رسیدن به مرحله ۲۶ پیکار استعدادیابی کامل میشه و شما با انجام دستورالعملی که توی مرحله ۲۶ می‌گیرین می‌تونین توی قرعه‌کشی شرکت کنین. یادتون باشه پیغام‌های شما رو Bot استخراج می‌کنه، پس طبق دقیقا اون چیزی که گفته شده عمل کنین. مثلا یه سری افراد سری پیش Hashtag درست رو نزدن و در قرعه‌کشی حتی وارد نشدن. و اما نفراتی که بتونن مرحله ۳۰ رو حل کنن شانسشون توی قرعه‌کشی بیشتر میشه (دو برابر، فقط شانس). از مرحله ۲۶ به بعد هیچ هینتی وجود نداره و باید با سرچ و مطالعه یا دانش قبلی پیکارها رو حل کنین.